```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
// initialize limitIntensity function
int limitIntensity(double value);
// initialize applyFilter function
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img);

int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, color_img;

    if ( argc != 2 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
    }

    /* set up structure for output color image */
    /* Note that the type is 'c' rather than 'g' */
    get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );

    // declare and initialize integer to store the dimension of the point spread function
    int PSF_dim = 9;

    // apply filter using applyFilter function as defined below main
    applyFilter(&color_img, &input_img);

    /* open color image file */
    if ( ( fp = fopen ( "filtered.tif", "wb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file color.tif\n");
        exit ( 1 );
    }

    /* write color image */
    if ( write_TIFF ( fp, &color_img ) ) {
        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
        exit ( 1 );
    }

    /* close color image file */
    fclose ( fp );

    /* de-allocate space which was used for the images */
    free_TIFF ( &(input_img) );
    free_TIFF ( &(color_img) );

    return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise, \n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}

// limitIntensity function definition
int limitIntensity(double inputValue) {
    // declare an integer variable newValue and initialize it to zero
    int newValue = 0;
    // if input value parameter is less than zero, assign new value to 0
    if (inputValue < 0) {
        newValue = 0;
    }
    // if input value parameter is greater than 255, assign new value to 255
    else if(inputValue > 255) {
        newValue = 255;
    }
    // otherwise, assign new value to the input value parameter re-cast as an integer
    else {
        newValue = (int)inputValue;
    }
    return newValue;
}

// applyFilter function definition
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img) {
    // declare and define image height and width based on input image TIFF struct methods
    int img_height = input_img->height;
    int img_width = input_img->width;
    // define array of size three to store RGB information for each pixel
    double plane[3];
    // for each pixel
    for (int i = 0; i < img_height; i++) {
        for (int j = 0; j < img_width; j++) {
            // for each plane in RGB pixel
            for (int k = 0; k < 3; k++) {
                // assign to plane the term that will always exist
                plane[k] = 0.01 * input_img->color[k][i][j];
                if (i > 0) {
                    // assign to plane the term that exists if i > 0
                    plane[k] += 0.9 * (input_img->color[k][i - 1][j]);
                }
                if (j > 0) {
                    // assign to plane the term that exists if j > 0
                    plane[k] += 0.9 * (input_img->color[k][i][j - 1]);
                }
                if (i > 0 && j > 0) {
                    // assign to plane the term that exists if i > 0 & j > 0
                    plane[k] += - 0.81 * (input_img->color[k][i - 1][j - 1]);
                }
                // populate output image method for color after calling limitIntensity function to ensure acceptable RGB values
                output_img->color[k][i][j] = limitIntensity(plane[k]);
            }
        }
    }
}
```